

# “NAAV” Programming Language

Group 17

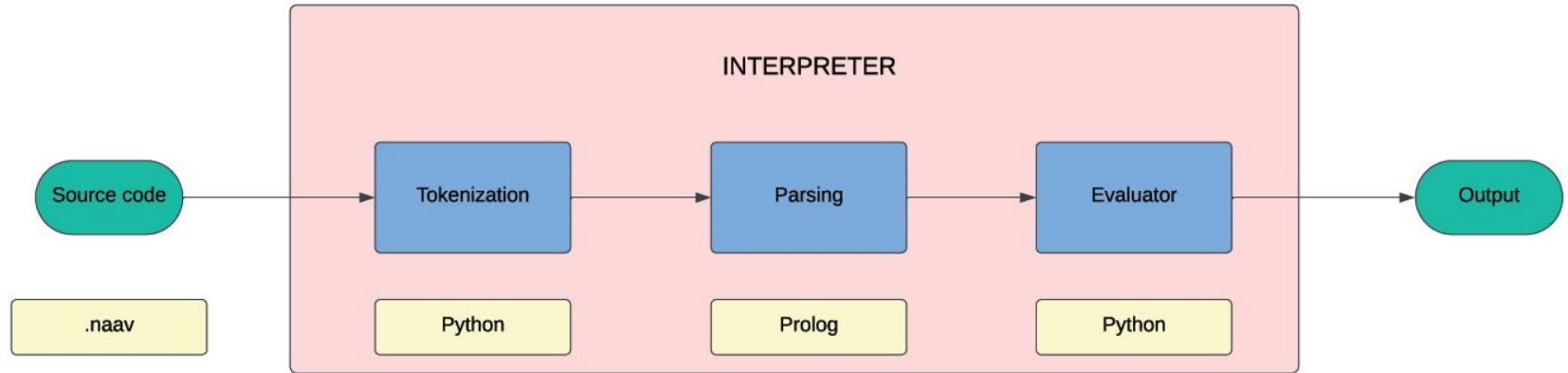
Team members: Atharva Date, Ansh Sharma, Vidya Rupak, Nisha Verma



# Introduction

- Name of the language ⇒ Initials of the group members' first name - **N**isha, **A**nsh, **A**tharva, **V**idya
- A versatile programming language with a user-friendly syntax ensuring code readability and ease of comprehension.
- It consists of familiar constructs like main method, declarations, commands and print statements.
- It supports primitive types such as boolean, numeric, and string, along with a rich set of operators and conditional and looping constructs to deal with complex logic efficiently.

# Interpreter



# Grammar

% Program rule

program --> block.

% Block rules

block --> [main], [{"", declarations, [";"], commands, printer, ["}"]].

block --> commands.

% Declarations rules

declarations --> declaration, [";"], declarations.

declarations --> declaration.

% Declaration rules

declaration --> var\_declaration | instant\_declaration.

var\_declaration --> type, id, assign\_op, value.

instant\_declaration --> type, id.

% Printer rule

printer --> [print], ["("], value, [")"], [";"].

printer --> [].

% Commands rules

commands --> command, [";"], commands.

commands --> command.

commands --> [].

# Grammar

% Command rules

command --> assign\_value.

command --> assign\_expression\_value.

command --> if\_else\_condition.

command --> ternary\_condition.

command --> increment\_command.

command --> decrement\_command.

command --> for\_loop.

command --> range\_loop.

command --> while\_loop.

command --> assign\_boolean\_value.

command --> assign\_boolean\_expression\_value.

assign\_value --> id, assign\_op, value.

assign\_expression\_value --> id, assign\_op, expression.

if\_else\_condition --> [if], ["("], condition, [")"], [{"{",  
block, ["}"], [else], [{"{", block, ["}"]].

ternary\_condition --> id, assign\_op, ["("], condition,  
[")"], ["?"], value, [":"], value.

increment\_command --> id, [++].

decrement\_command --> id, [--].

for\_loop --> [for], ["("], declaration, [;], condition, [;],  
increment\_expression, [")"], [{"{", block, ["}"]].

for\_loop --> [for], ["("], declaration, [;], condition, [;],  
decrement\_expression, [")"], [{"{", block, ["}"]].

range\_loop --> [for], type, id, [in], [range], ["("], value,  
[",", value, [")"], [{"{", block, ["}"]].

while\_loop --> [while], ["("], condition, [")"], [{"{", block,  
["}"]].

assign\_boolean\_value --> id, assign\_op, value.

assign\_boolean\_expression\_value --> id, assign\_op,  
boolean\_expression.

# Grammar

% Expression rules

expression --> arithmetic\_expression.

arithmetic\_expression --> term.

arithmetic\_expression --> term, arithmetic\_operator,

arithmetic\_expression\_higher\_precedence.

arithmetic\_expression\_higher\_precedence --> term, {

precedence(Op, Precedence), Precedence > 0 },

arithmetic\_operator,

arithmetic\_expression\_higher\_precedence.

arithmetic\_expression\_higher\_precedence --> term.

% Term rules

term --> factor | factor, arithmetic\_operator, term.

% Factor rules

factor --> value.

factor --> ["("], arithmetic\_expression, [")"].

% Arithmetic operator rules

arithmetic\_operator --> ['/' ] | ['\*'] | ['+'] | ['-'].

% Type rules

type --> [num] | [bool] | [string].

% ID rules

id --> identifier.

# Grammar

## % Value rules

value --> number | string | boolean | identifier.

number --> [X], { number(X) }.

string --> [X], { string(X) }.

boolean --> [T], { member(T, ['T']) } | [F], { member(F, ['F']) }.

identifier --> [X], { member(X,  
[a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]) } |  
[X], { atom(X), \+ member(X,  
[a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]) }.

## % Assign operator rules

assign\_op --> [=].

## % Comparison operator rules

comparison\_operator --> [>] | [<] | [>=] | [<=] | [==] |  
[<>].

## % Boolean operator rules

boolean\_operator --> [and] | [or] | [not].

## % Condition rules

condition --> value, comparison\_operator, value |  
boolean\_expression.

## % Boolean expression rules

boolean\_expression --> value.

boolean\_expression --> value, boolean\_operator, value.

boolean\_expression --> ["("], condition, [")"],

boolean\_operator, ["("], condition, [")"].

boolean\_expression --> ["("], condition, [")"],

boolean\_operator, value.

boolean\_expression --> value, boolean\_operator, ["("],  
condition, [")"].

# Execution

- A python file called 'executor' which contains the three modules in the interpreter is executed to run the NAAV code.
- The executor file is invoked via the command line, with the filepath of the source code provided as an argument, initiating the execution process.
- The results of the code execution are presented on the command line interface, providing users with immediate feedback and visibility into the program's behavior.



# Tools and Technologies

- System: Windows OS, Mac OS
- Languages: Python and Prolog
- Tools: PyCharm, SWI-Prolog, Google Drive, VSCode
- Libraries: Ply, PySwip

# Language Demo

THANK YOU